Computer Graphics Lab 5 Report

一、实验内容

光线追踪

二、测试环境

OS: OpenSUSE Leap 42.2

CPU: Intel® Core™ i7-6500U @ 2.60GHz

RAM: 8GB DDR4 2133MHz

Programming Language: GCC-C++ 6.3.0

三、操作说明

程序自动渲染一张预先设定的四个球体的画面

四、实现过程

1. 对屏幕上每个点利用光线追踪计算颜色值

```
for (int i = 0; i < MAX_SIZE; i++)
        for (int j = 0; j < MAX_SIZE; j++) {
                float x = (i - MAX_SIZE * 0.5) / (MAX_SIZE * 0.5);
                float y = (j - MAX_SIZE * 0.5) / (MAX_SIZE * 0.5);

                Vector3f position(x, y, 0.0);

                Ray ray(eye, (position - eye).Unitization());

                RayTracing(ray, 1.0f, color[i][j]);
        }
```

2. 光线追踪过程：

光线追踪函数有射线、权值和待计算颜色三个参数。

（1）解析几何方法用直线和球体求交，求出所有交点，返回距离视点最近的。并判断光线经过的是球面还是空间，从而确定法向应取内法向或是外法向。

```
Point Cross(Ray ray) {
        vector<Point> cross;

        for (int i = 0; i < MAX_SHPERE; i++) {
                Sphere & sph = sphere[i];

                Vector3f middle = ray.dir * ((sph.center - ray.start) * ray.dir) + ray.start;

                float dis = (sph.center - middle).Modulus();

                if (dis >= sph.radius)
                        continue;
                else {
                        Vector3f delta = ray.dir * sqrt(pow(sph.radius, 2.0) - pow(dis, 2.0));

                        if ((middle - delta - sph.center) * ray.dir < FLOAT_EPS)
                                cross.push_back(Point(middle - delta, ((middle - delta) - sph.center).Unitization()));
                        else
                                cross.push_back(Point(middle - delta, (sph.center - (middle - delta)).Unitization()));

                        if ((middle + delta - sph.center) * ray.dir < FLOAT_EPS)
                                cross.push_back(Point(middle + delta, ((middle + delta) - sph.center).Unitization()));
                        else
                                cross.push_back(Point(middle + delta, (sph.center - (middle + delta)).Unitization()));
                }
        }

        sort(cross.begin(), cross.end(), [=] (auto & a, auto & b) {
                        return (a.position - ray.start) * ray.dir < (b.position - ray.start) * ray.dir; });

        for (auto & tempPoint : cross)
                if ((tempPoint.position - ray.start) * ray.dir > FLOAT_EPS)
                        return tempPoint;

        return Point(Vector3f(0.0, 0.0, 1.0), Vector3f());
}
```

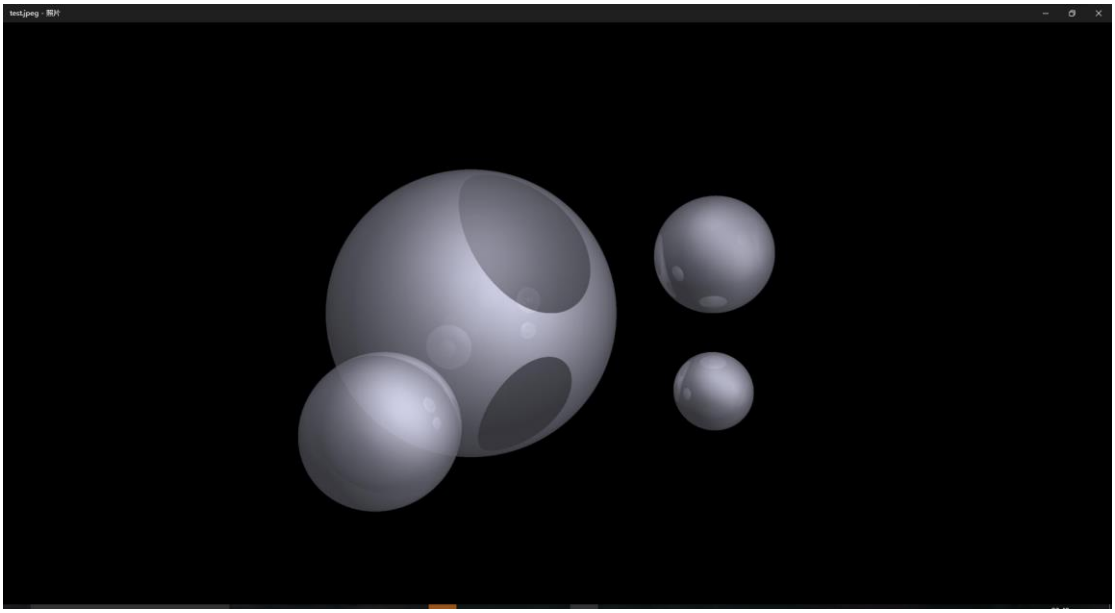
（2）对每个光源计算该点的光亮度值，进一步算出颜色值。这里要判断点到光源是否被遮挡，也就是该点是否是光源到该方向的最近交点。

```
float lambda = IA * KA;

for (int i = 0; i < MAX_LIGHT; i++) {
        Ray ray(light[i], (cross.position - light[i]).Unitization());

        if (Cross(ray) != cross) continue;

        Vector3f L = (light[i] - cross.position).Unitization();
        Vector3f N = cross.normal;
        Vector3f H = (L + (eye - cross.position).Unitization()).Unitization();

        lambda += IP * (KD * (L * N) + KS * pow(H * N, 8.0));
}

return LIGHT * lambda;
```

（3）计算反射角、透射角，递归地计算反射光线和透射光线。

```
Ray FuncRay_S(Ray ray, Point cross) {
        Vector3f dir = ray.dir - cross.normal * (ray.dir * cross.normal) * 2.0;
        return Ray(cross.position, dir);
}

Ray FuncRay_T(Ray ray, Point cross) {
        Vector3f dir = ray.dir;
        return Ray(cross.position, dir);
}
```

五、结果示例



六、小结

实现了空间的光线追踪算法，可以模拟反射、透射等光学现象。